УДК 004

Сравнение монолитной и микросервисной архитектуры приложения с использованием JSON Web Tokens

Голубь Илья Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Магистрант

Аннотация
В данной работе приведено сравнение монолитной и микросервисной архитектуры приложения. Так же, рассматривается микросервисная архитектура с использованием JWT аутенфикации, пример создания JWT токена и способ увеличения уровня безопасности.
Ключевые слова: Микросервис, JWT, монолит, архитектура приложений, языки программирования

Comparison of monolithic and microservice application architecture using JSON Web Tokens

Golub Ilya Sergeevich
Sholom-Aleichem Priamursky State University
Undergraduate

Abstract
This paper compares applications of monolithic and microservice architecture. Architecture using JWT authentication, an example of creating JWT current and a way to increase security.
Keywords: Microservice, JWT, monolith, application architecture, programming languages

1. Введение

Тема микро сервисной архитектуры очень актуальна в данный момент, большинство организаций переводит свои большие, монолитные приложения и веб-сервисы, к микро сервисам, путем деления. Микросервисные приложения легче поддерживать, развивать. Но, что бы на разработку и поддержку данных сервисов уходило меньше рабочего времени разработчика нужно, что бы сборкой проекта, созданием резервной копии и развертыванием проекта в тестовой и продуктивной среде занималось средство ci cd. Так же обновление и доработка сервисов происходи вне зависимости друг от друга, что означает непрерывную работу и не значительное влияние друг на друга, оказанное обновлением одного или всех сервисов. Так же повышение безопасности а так же улучшение споособов общения сервисов друг с другом достигается за счет использования JSON Web Token (JWT) — это JSON объект, который определен в открытом

стандарте RFC 7519. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок (header) с общей информацией по токену, полезные данные (payload), такие как id пользователя, его роль и т.д. и подписи (signature).

1.1 Обзор статей

В статье А.И. Иванова исследована проблема использования сервис-ориентированной архитектуры для средних предприятий. Предложено использование архитектуры микросервисов, как решение данной проблемы. Выявлены достоинства и недостатки этой архитектуры.[6]

Так же существует микросервис VMware, который запотентовали, областью применения данного микросервиса является управление виртуализацией VMware vSphere. Целью разработки программы является реализация микросервиса, предоставляющего HTTP RESTful API для взаимодействия с виртуальными машинами VMware vSphere. Такой микросервис может осуществлять все основные высокоуровневые операции над виртуальной машиной и может быть использован другими программными средствами. Функциональные возможности: управление виртуальными машинами VMware vSphere; возможность расширения списка операций; HTTP RESTful API [7.

Микросервисы не привязанные к какому либо языку, по этому осуществлять разработку программ на микросервисной архитектуре можно на любом языке, например в статье В.П. Замышляева рассматривается создание микросервиса с архитектурой REST API при помощи фреймворка Flask на языке программирования Python [5].

В статье Л. Хавьер, С. Изаскун, К-П. Рикардо, Э. Кристофер, рассказывается о развитии технологий микросервисов, которое сегодня происходит весьма стремительно, однако для микросервисов требуется культура DevOps, поэтому и начинать нужно с нее, что быстро станет приносить пользу благодаря интеграции процессов разработки и эксплуатации [10].

В работе Г. А. Опарин, В. Г. Богданова, А. А. Пашинин рассматриваются вопросы применения микросервисного подхода для конструирования и поддержки функционирования распределенных решателей вычислительных задач на модели предметной области[8].

Н. Д. Осипова занималась рассмотрением плюсов и минусов монолитного и микросервисного подходов к разработке программного обеспечения и критерии принятия решения о переходе на микросервисную архитектуру на примере разработанного микросервиса интеграции[9].

В работе N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina рассматривается история архитектуры программного обеспечения, причины, которые привели вначале к распространению объектов и услуг, а затем микросервисы. Наконец, открытые проблемы и будущие проблемы представлены. Этот опрос в

первую очередь адресован новичкам в дисциплине, предлагая академическую точку зрения на эту тему. Кроме того, мы исследуем некоторые практические проблемы и указываем несколько возможных решений[2].

В своей статье A. Balalaie, A. Heydarnoori, P. Jamshidi рассматривают опыт и уроки, извлеченные в ходе поэтапной миграции и архитектурного рефакторинга коммерческого мобильного бэкенда в качестве сервиса для архитектуры микросервисов. Это объясняет, как исследователи приняли DevOps и как это способствовало плавной миграции[1].

В работе C. Pahl, P. Jamshidi рассматриваются микросервисы, которые недавно стали архитектурным стилем, рассматривающим способы создания, управления и развития. Архитектура из небольших автономных модулей. В частности, в облаке подход к архитектуре микросервисов представляется идеальным дополнением контейнерных технологий на уровне PaaS.в настоящее время нет вторичного исследования, чтобы закрепить это исследование. Мы стремимся здесь идентифицировать, таксономически классифицировать и систематически сравнивать существующие исследовательские центры по микросервисам и их применение в облаке. Мы провели систематическое картографическое исследование 21 отобранных исследований, опубликованных за последние два года до конца 2015 года с момента появления модели микросервисов. Мы классифицировали и сравнивали выбранные исследования основанный на структуре характеристики. Это приводит к обсуждению согласованных и возникающих проблем в архитектурном стиле микросервисов, позиционируя его в контексте непрерывного развития, но также приближая его к облачным и контейнерным технологиям[3].

В статье G.Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, A. Edmonds исследуется эффективный запуск приложений в облаке, который требует гораздо большего, чем развертывание программного обеспечения на виртуальных машинах. Облачные приложения должны постоянно управляться: 1) адаптировать свои ресурсы к входящей нагрузке и 2) противостоять временным сбоям репликации и перезапуска компонентов, чтобы обеспечить отказоустойчивость ненадежной инфраструктуры. Непрерывное управление отслеживает метрики приложений и инфраструктуры, обеспечивая автоматическую и оперативную реакцию на сбои (управление работоспособностью) и изменение условий окружающей среды (автоматическое масштабирование), сводя к минимуму вмешательство человека. В существующей практике функции управления предоставляются как инфраструктурные или сторонние сервисы. В обоих случаях они являются внешними по отношению к развертыванию приложения. Мы утверждаем, что этот подход имеет внутренние ограничения, а именно то, что отделение функций управления от приложения предотвращает их естественное масштабирование с приложением и требует дополнительного кода управления и вмешательства человека. Более того, использование услуг провайдера инфраструктуры для функций управления приводит к тому, что вендор эффективно блокируется, что не позволяет

облачным приложениям адаптироваться и работать в наиболее эффективном облаке для работы. В этой позиции мы предлагаем новую архитектуру, которая обеспечивает масштабируемое и гибкое самостоятельное управление приложениями микросервисов в облаке[4].

1.2 Цель исследования

Цель данного исследования — доказать что использование JWT увеличивает безопасность микросервисной архитектуры.

2. Методы и материалы исследования

Сравнение обычного общения клиентской части приложения с его серверной частью. В качестве материала исследования возьмем пример монолитной архитектуры приложения на ASP.NET core и его микросервисного аналога. Монолитная архитектура приложения будет выглядеть как на рис.1.

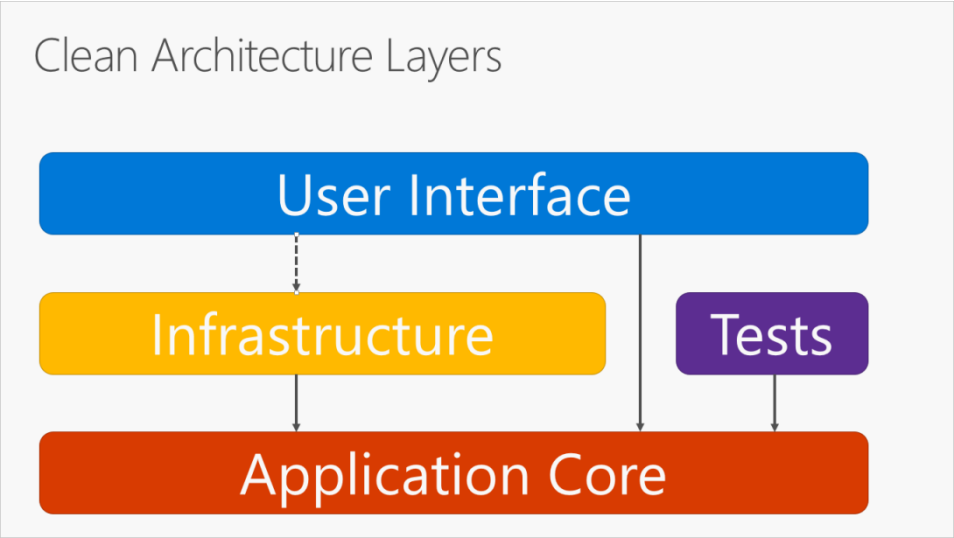

Рис. 1. Монолитное приложение

Как видно на рисунке каждая часть проекта, а именно User Interface(UI) - пользовательский интерфейс, infrastructure – инфраструктура, test – тестирование, а так же Application core – ядро приложения находятся в прямой зависимости. Каждая из частей проекта общается с ядром, что может привести к проблемам в безопасности, а так же к сложностям в доработке проекта. Получив от пользователя данные, UI идет в инфраструктурные решения и даже на прямую в ядро, что можно повлечь за собой внедрение инъекций со стороны пользователя, передачу не желательного кода через встраивание в запросы своего кода от пользователя. Так же такие зависимости при разработке влекут за собой постоянные изменения полной кодовой базы на всех уровнях потому, что каждый уровень зависит от остальных.

Так же нужно обратите внимание на то, что сплошные стрелки соответствуют зависимостям времени компиляции, а пунктирные —

зависимостям, существующим только во время выполнения. В рамках чистой архитектуры слой пользовательского интерфейса работает с интерфейсами, которые определены в ядре приложения во время компиляции. Во время выполнения эти типы реализации необходимы для выполнения приложения, поэтому они должны существовать и быть привязаны к интерфейсам ядра приложения посредством внедрения зависимостей. На рисунке 2 показано более подробное представление данной архитектуры.

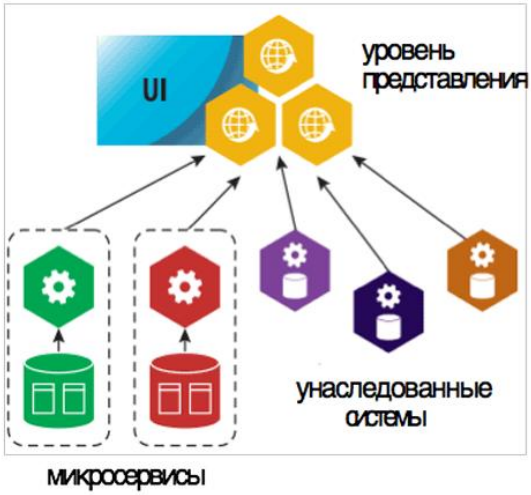

Рис. 2. Микросервисная архитектура

Рассмотрим микросервисную архитектуру приложения. Как видно из рисунка 2 то Микросервисы, унаследованные системы отдают данные в UI, при этом не зависят друг от друга и следовательно, внедрение инъекций кода от пользователя, которые могу повлиять на ядро микросервисов становится маловероятным.

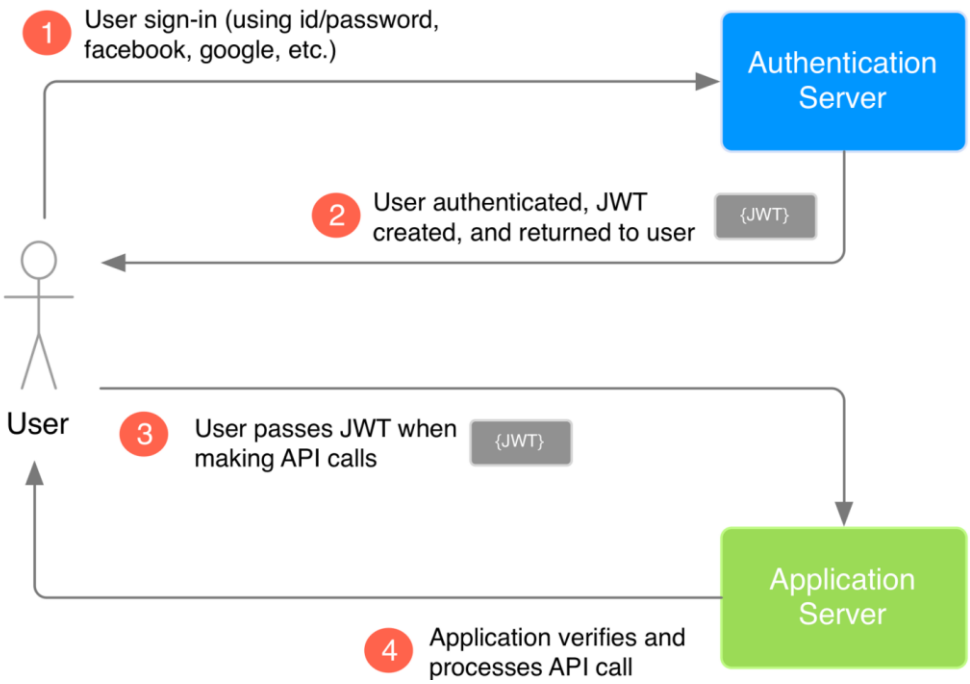

Рис. 3. Микросервисная архитектура с использованием JWT токена.

Приложение использует JWT для проверки аутентификации пользователя следующим образом:

Сначала пользователь заходит на сервер авторизации с помощью авторотационного ключа (это может быть пара логин/пароль, либо Facebook ключ, либо Google ключ, либо ключ от другой учетной записи).

Затем сервер авторизации создает JWT и отправляет его пользователю.

Когда пользователь делает запрос к API приложения, он добавляет к нему полученный ранее JWT.

Когда пользователь делает API запрос, приложение может проверить по переданному с запросом JWT что это за пользователь, его права, а так же уровень его доступа. В этой схеме сервер приложения сконфигурирован так, что сможет проверить, является ли входящий JWT именно тем, что был создан сервером аутентификации.

JWT токен разделен на три части:

Header – заголовок;

Payload – полезные данные;

Signature – подпись

Заголовок содержит в себе иформации о том, как должна вычисляться JWT подпись. Заголовок передается в виде JSON объекта и выглядит следующим образом:

header = { "alg": "HS256", "typ": "JWT"}

В поле type мы указываем что это JWT токен. Поле alg определяет алгоритм хеширования. Данный алгоритм будет использоваться при создании подписи. В данном случае HS256=SHA256, для его вычисления нужен лишь один секретный ключ. Так же возможно использование и другого алгоритма, а именно RS256 – в отличии от предыдущего, он является ассиметричным и создает пару ключей: Приватный и публичный. Приватный ключ нужен для создания подписи, а публичный ключ используется для проверки подлинности данной подписи, это еще одна, дополнительная мера безопасности.

Полезные данные (PayLoad) – это полезные данные, хранящиеся внутри JWT токена. Так же, эти данные называют JWT claims (заявки). В данном примере рассматривается сервер аутенфикации, который создает JWT с информацией о пользователе – userId.

payload = { "userId": "b08f86af-35da-48f2-8fab-cef3904660bd" }

В данном примере используется всего один claim, но их может быть не ограниченное количество. Так же существует некоторый стандартный список claim’ов основные из них:

iss (issuer) — определяет приложение, из которого отправляется токен.

sub (subject) — определяет тему токена.

exp (expiration time) — время жизни токена.

Этот список может быть полезен при создании JWT, но данные поля не являются обязательными. Чем больше claim’ов мы создаем, тем больше по размеру получается сам токен, что может негативно повлиять на

производительность и вызвать не желаемые задержки во время взаимодействия с сервером.

Остается создать подпись:

```
const SECRET_KEY = 'cAtwa1kkEy'
const unsignedToken = base64urlEncode(header) + '.' + base64urlEncode(payload)
const signature = HMAC-SHA256(unsignedToken, SECRET_KEY)
```

Алгоритм base64url кодирует заголовок и полезные данные. Алгоритм соединяет закодированные данные через точку. Далее, полученная строка хешируется алгоритмом, заданным в хедере на основе нашего секретного ключа.

```
// header eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
// payload eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ
// signature -xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

На последнем этапе происходит объединение всех составляющих в токен JWT.

```
const token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.' + encodeBase64Url(signature)
// JWT Token
// eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

Само по себе использование JWT не скрывает и не маскирует данные автоматически. Смысл использования JWT – это проверка, что отправленные данные были отправлены из авторизованного источника. Данные были закодированы и подписаны. Цель кодирования – преобразование структуры.

Для увеличения безопасности нужно использовать RT токен. Токен обновлениф (Refresh token) – это токен, выполняющий лишь одну задачу – получение нового токена доступа. Данный токен является одноразовым и нужен он, что бы обновлять JWT токен следовательно, если JWT токен был скомпрометирован, запуская RT токен мы получаем новый JWT чем делаем старый токен не действительным.

3. Результат исследования

Получается, что при использовании обычной аутенфикации, пользователь сразу отправляет данные на конечные URL сервера постоянно. А при использовании JWT токена пользователь проходит только по одному URL который выдает ему токен и авторизует его.

4. Дискуссия

Данная работа может помочь программистам, которые думают над использование JWT принять решение, так же, данная статья рассматривает несколько походов к построению приложений, микросервисная архитектура и монолитная архитектура, что так же будет полезно для разработчика.

Выводы

Результатом работы стало понимание того, что при использовании JWT с RT токеном безопасность повышается, т.к. по URL ходят закодированные данные, время жизни этих данных ограниченно и они постоянно обновляются. У Пользователя нет конечных URL сервера.

Библиографический список

1. Balalaie A., Heydarnoori A., Jamshidi P. Microservices architecture enables devops: Migration to a cloud-native architecture //Ieee Software. 2016. Т. 33. №. 3. С. 42-52.
2. Dragoni N. et al. Microservices: yesterday, today, and tomorrow //Present and ulterior software engineering. – Springer, Cham, 2017. С. 195-216.
3. Pahl C., Jamshidi P. Microservices: A Systematic Mapping Study //CLOSER (1). 2016. С. 137-146.
4. Toffetti G. et al. An architecture for self-managing microservices //Proceedings of the 1st International Workshop on Automated Incident Management in Cloud. ACM, 2015. С. 19-24.
5. Замышляев В. П., Попок Л. Е. создание rest api микросервиса с использованием flask // Информационное общество: современное состояние и перспективы развития сборник материалов XII международного студенческого форума. - Краснодар: ФГБОУ ВО «Кубанский государственный аграрный университет имени И. Т. Трубилина», 2019.
6. Иванов А. И. Архитектура микросервисов для предприятий малого и среднего бизнеса, как замена сервис-ориентированной архитектуры // Инновационные подходы к решению технико-экономических проблем. 2015. №2.
7. Микросервис управления виртуальными машинами vmware vsphere (vm_restfullapi v1.10): свидетельство о государственной регистрации программы для ЭВМ / Дворников В. С., Радько В. А., Шаповалов В. Н. – № 2018666988 Дата регистрации: 30.10.2018
8. Опарин Г. А., Богданова В. Г., Пашинин А. А. Микросервисы как фундаментальная основа распределенного сборочного программирования // Информационные технологии в науке, образовании и управлении. 2018. №6.
9. Осипова Н. Д. Разработка микросервиса интеграции системы самообслуживания абонентов сотовой связи и центра нотификаций в рамках перехода от монолитной архитектуры приложения к микросервисной // Естественные и математические науки в современном

МИРЕ. 2017. №51.
10. Dragoni N. et al. Microservices: yesterday, today, and tomorrow //Present and ulterior software engineering. – Springer, Cham, 2017. С. 195-216.
11. Хавьер Л. , Изаскун С., Рикардо К-П., Кристофер Э. Микросервисы // открытые системы. Субд. 2018. №3.